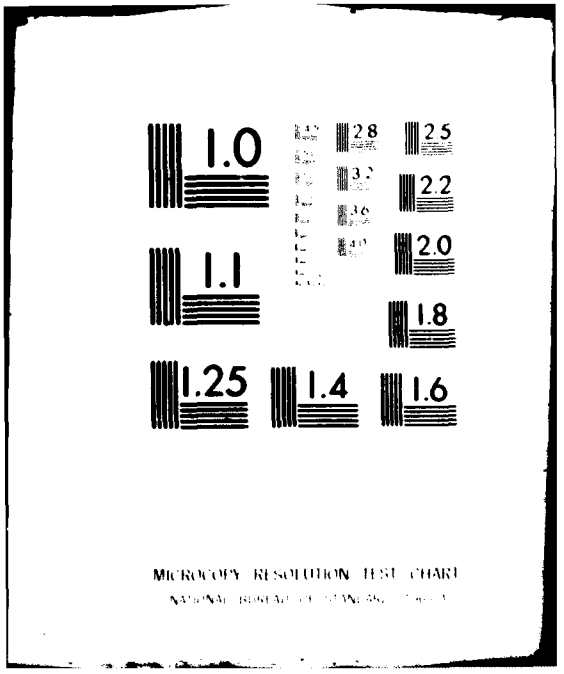
AD-A090 759 GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION A--ETC F/G 9/2
A SYSTEM ARCHITECTURE TO SUPPORT A VERIFIABLY SECURE MULTILEVEL--ETC(U)
JUN 80 G I DAVIDA, R A DEMILO, R J LIPTON N00014-79-C-0873
UNCLASSIFIED GIT-ICS-80/05 NL

END
DATE FILMED
11-80
DTIC


1.0
2.8
2.5
3.2
2.2
3.6
2.0
4.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART

437 LEVEL II (12)

AD A090759

School of

Information and Computer Science

GEORGIA INSTITUTE OF TECHNOLOGY

80 8 4 035

LEVEL II

(12)

(14) GIT-ICS-80/05

(6) A System Architecture to Support A Verifiably Secure Multilevel Security System (Extended Abstract)

(10) George I. Davida*
Richard A. DeMillo**
Richard J. Lipton***

Contract N00014-79-C-0873 (15)

(12) 7

(11) June 1980

* University of Wisconsin, Milwaukee
** Georgia Institute of Technology
*** University of California, Berkeley
(Currently at Princeton University)

A System Architecture To Support A Verifiably Secure Multilevel Security System[†]

(Extended Abstract)

George I. Davida
Department of Electical Engineering and Computer Science
University of Wisconsin, Milwaukee
Milwaukee, Wisconsin 53201

Richard A. DeMillo
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

Richard J. Lipton*
Department of EE and Computer Science
University of California, Berkeley
Berkeley, California 95430

*Current Address
Department of EE and Computer Science
Princeton University
Princeton, N.J. 08540

[†]This research was supported in part by the Georgia Tech Distributed Computing Project which is funded by ONR Grant No. N00014-79-0873 and by AFOSR grant to the Charles Stark Draper Laboratories.

A SYSTEM ARCHITECTURE TO SUPPORT A VERIFIABLY SECURE MULTILEVEL SECURITY SYSTEM*
(Extended Abstract)

George I. Davida
University of Wisconsin
Milwaukee, Wisconsin

Richard A. DeMillo
Georgia Institute of Technology
Atlanta, Georgia

Richard J. Lipton
University of California
Berkeley, California

1. Introduction

Technology that allows significant sharing of computer resources carries with it an increased responsibility to protect these resources from unauthorized, malicious, irresponsible, or unintended use or disclosure. The years have seen a progression of increasingly sensitive information made available in increasingly less supervised modes to a variety of users. Commercial users routinely store valuable financial information and conduct cashless transactions electronically. University professors maintain class grading forms and examinations on departmental computers. Government agencies keep extensive databases of sensitive information regarding employees, foreign nationals, U.S. citizens. The military and intelligence communities continue to press for more powerful techniques to enhance their information gathering and processing capabilities. In spite of the clear need for guarantees of security, all practical schemes to protect information stored or manipulated by such systems are either seriously flawed or reduce ultimately to a collection of physical security protocols (see [1] for an overview of the state of the art).

These issues have their clearest expression in the area of multilevel security. The basic multilevel security problem is to share a collection of documents among groups of users who have differing "clearances" regarding the information embodied in the documents. The simplest of the security problems which can be formulated in this setting is that of insuring the *-property [2]: information that flows from a user of a clearance level A to one of clearance level B, where A is "higher" than B does not allow the users at level B to have access to information to which they are not entitled. This is an obvious generalization of the military confidential/secret/top secret classification scheme and occurs as a subproblem in most computer related security problems. It is this problem which we will attack in the sequel.

The crux of the problem seems to be this: what software shall be trusted by the security system? If the subsystems that process requests for reading or writing information, for authenticating the identity and authorization of users, and for carrying out the related data processing functions are all equally trustworthy then it is possible for clever users to make trusted components betray components which use them and thus compromise the system [3]. A commonly proposed [4] solution is to severely restrict the amount of computer code that has to be trusted. This amounts to a small operating system, called a kernel through which all secure transactions must be funnelled and which will enforce the security of the system. If the kernel can be relied upon to behave as it should,

*This research was supported in part by the Georgia Tech Distributed Computing Project which is funded by ONR Grant No. N00014-79-0873 and by AFOSR grant to the Charles Stark Draper Laboratories. Authors' current addresses: G. Davida, University of Wisconsin, Milwaukee, Milwaukee, WI 53201, R. DeMillo, School of Information & Computer Science, Georgia Institute of Technology, Atlanta, Ga. 30332, R. Lipton, Department of EE and Computer Science, University of CA, Berkeley, Berkeley, CA 94207.

then the security problem is solved! The problem thus reduces to determining the trustworthiness of a seemingly well-defined program, the kernel.

Some researchers (cf. [4]) have sought techniques capable of inducing absolute confidence in the trustworthiness of the kernel, usually by a valid mathematical proof that the kernel performs as specified; this is sometimes called a verification of the kernel. A great deal of effort has been expended along these lines. While the prospects for the success of this approach remain uncertain (for differing views on the matter compare [5] and [6]), the success of the verified kernel approach may not be particularly relevant for the ultimate goal: to obtain with reasonable (and hence finite) cost usuable but acceptably secure computer systems. A point that has become somewhat clouded in recent years is that while the success of the verified kernel approach is sufficient for security, it is not necessary! The theme of this note is that there are other approaches to the security problem, approaches which require neither the concept of a kernel nor the verification of a single line of code.

We will present below a sketch of three system organizations which achieve varying levels of security at varying costs. None of these methods depend on technological breakthrough, although all of them depend heavily on the current trend of decreasing hardware costs. Most importantly, we do not require the system to trust any software at all. There are, however, trusted components; the components in which the system must place its trust all achieve their security by processes which have intrinsic value outside the system. This means that communities of scientists are inclined to study these processes regardless of whether or not a particular security system is actually built. Furthermore it is the processes whose properties are established by such "socialization" that determine directly the trustworthiness of the components. This is the key aspect of socially acceptable argumentation discussed in [6]. Components whose properties are established in this way are as trustworthy as, say, mathematical theorems whose proofs have been subjected to the same sort of scrutiny. Therefore, we feel justified in calling such schemes "verifiably secure."

Let's summarize. The approach to be outlined here is fundamentally different from the verified kernel approach. We assume that

(*) no software will be trusted to yield the security of the total system.

Thus, we reject completely the idea of verifying any code! Instead, we intend to rely on certain hardware components. Our second assumption is

(**) the security of the total system derives from a few very simple hardware devices by socially acceptable arguments.

It is critical to note at the outset that we will not simply propose that a security kernel be placed on a chip; this is the same as the verified kernel approach. Of course, our system will have software -- in all likelihood, a great deal of it! The role of the software will, however, be to provide services to users; the role of software is therefore effectively decoupled from its utility in supporting or breaking the security of the system. This is an important distinction when viewed in the following context: when the software also mediates the security mechanisms, the services supplied by the software (i.e., its "functionality") directly affect the level of security provided by the system. In particular, a system can be made perfectly secure if it provides no services and therefore has no users. Higher functionality systems are more vulnerable by their nature.

We will present three approaches to the multilevel problem. Approach A will be described in Section 3, while approaches B and C will be described in Section 4. The following table summarizes the essential characteristics of these approaches.

Approach	Cost[+]	Need to Trust	Level of Security Provided
A	N**2	simple physics and physical protocols	extremely secure
B	N	simple physics and protocols, encryption, simple logic	requires secure cryptosystems, but can leak a single bit
C	1	simple physics and protocols, encryption, some logic	requires secure cryptosystems, complex protocols that can change accesses in software; still only leak a single bit

2. The *-Property

As we discussed above, we will deal exclusively with the multilevel security problem. The interested reader should consult [7] for a full discussion and for motivating problems. We assume that the system is static and given by a directed acyclic graph S with nodes sl,...,sn. Each si represents a security level, that is, a user or group of users at the same "clearance" level who have indistinguishable need to know, or other similar authorization. The arcs of the graph S relate the security levels, so that we write

si --> sj

when si has a higher security level than sj. The *-property states that information can flow from si to sj only when there is a directed path from sj to si. While there are more complex issues that can be addressed (e.g., the confinement problem [8]), the basic military problem is captured by the *-property requirement. It is well-known that even the simple *-property is difficult to achieve; in a system with high functionality information can flow between users in very subtle ways.

3. Approach A

Our first "solution" is based on one hardware device: the one-way link. That is, we assert the existence of a physical device which allows information to flow in one direction exclusively. Our confidence in the security of the overall system is limited only by our confidence in the unidirectional nature of the link. That is in stark contrast to the "trusted software" situation; if we had instead asserted the existence of a piece of code through which information flowing in one direction could be distinguished from information flowing in another direction, the confidence in system security would be limited, first, by our confidence in the software (this level of confidence to be established by a proof of correctness or some other technique) and, second, by our confidence that a sufficiently clever adversary cannot exploit the inherent functionality of the software to effect a "reverse" transmission (i.e., cannot "cheat" the underlying security model). The hardware and software systems are really not the same at all: to fool the software, it is only necessary to avoid violating specifications, but to fool the hardware, it is necessary to violate laws of physics!

Standard TTL twisted pairs come close to the ideal one-way link, even though information may be leaked in several ways. Fiber optic links give an essentially perfect solution (see Figure 1). USER-1 cannot even tell whether or not USER-2 exists, and the only way that USER-2 can send information to USER-1 is to actually swap the properties of the emitter and the sensor. In other words, he must physically convert a light sensitive device to a light emitting device! This is a sufficiently blatant act that we assume physical monitoring procedures will alert the security manager if the roles of the sending-receiving devices are ever reversed.

[+] System cost is measured as a function of the number of security levels in the system. Intuitively, it is the number of copies of system files, pages, or documents which must be maintained. If each accessible object is maintained on its own device, then cost is related to the number of disk drives, for example.

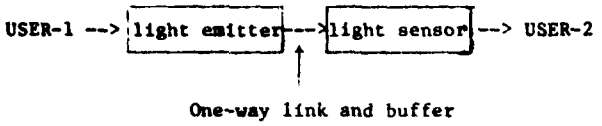


Figure 1. A One-Way Optical Link

The "buffer" in Figure 1 is intended to be a hardware device to aid in the high-volume transfer of data from USER-1 to USER-2; in particular, such a buffer may be needed to insure that timing signals need not be passed from USER-2 to USER-1 or to avoid redundancy in the transmission. In applications, the buffer may be a small computer which is not accessible to users, provides no services and is never reprogrammed. For added security it may be physically sealed. It is not required that the buffer be trusted as long as it cannot be accessed from outside. At worst, the buffer can fail to send the proper information to USER-2; this however does not violate the unidirectionality of the link. The buffer can retain the information, but if it is sealed, then no unauthorized access is possible.

Even with the relatively high degree of confidence we have in the nature of the one-way link, sending large volumes of data over the link is a possible weakness. In solution B below, however, the one-way link carries only a few bytes at a time, precluding the need for complex buffering and mitigating against casual eavesdropping.

Now, assuming one-way links does very little good if there are hidden (i.e., software) paths for the information, so part of our assumption is that information may only flow between levels through one of these links. We will show very shortly that this does not necessarily imply separate storage facilities for all levels. It does imply, however, that the processing stages of each security level should have no interaction with the processing at any other level. The only way to insure this without trusting any software is to assume that each level has its own "computer". A computer can be an IMB 370, a DEC PDP-11/45, or a TRS-80. The choice of computers at each level is irrelevant for security purposes; the choice is a function of the processing needs at that level and by economics. However, this is the age of cheap personal computers. Processors are becoming very inexpensive. Therefore, we believe that allocating one processor to each security level is a quite reasonable decision. The advantage to the user is that the user may run whatever system software he wants at his level. He is not bound to the restricted functionality of a "secure" operating system, for instance.

It should be noted that providing a separate processor for each security level, is not the same as providing separate information systems for each level, which of course solves the problem completely. Information can still flow between the levels, and in approaches B and C, we will even allow information from the various levels to be housed in the same physical devices!

We now turn to the design. Let us assume two levels, say s1 and s2. We assume that each of these levels corresponds to an entire computer system. We must complete the design by supplying a way for s2 to read any of s1's files and at the same time avoid any information flow from s2 to s1. Assume that s1 and s2 use a disk for storage of their files. We use the term "disk" generically for any mass storage device; the solutions here and in the sequel do not depend on the kind of storage technology used.

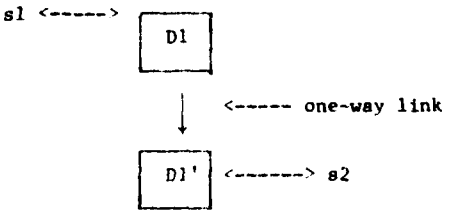


Figure 2. Shadow Disk for s1

Let D1 be the disk for s1. We assume that there is a one-way link from D1 to another disk, called the shadow disk for s1, D1' (see Figure 2). All writes of s1 to D1 are sent down the one-way link to be performed on D1' as well. At all times D1' is a "shadow" copy of D1. Whenever s2 accesses the files of s1, the access is through s1's shadow disk, only.

The design is obviously secure. If the one-

way links work then it is impossible for information to flow from s2 to s1. As simple as the design is, it illustrates our approach. Security stems from the physical configuration of the system and not from complex software.

Two additional problems must be solved. Observe that D1' may lose information during write commands due to rotational delay. There are two solutions. We can either require that D1 is a slower disk than D1' or, alternatively, D1 may be forced to run slower than D1'. In any case, we cannot allow even a one bit line from D1' to D1 to perform this synchronization. Therefore, we must pay a worstcase cost everywhere to avoid dangerous information flow. The second problem, concerns a possible hidden channel for passing information from s2 to s1. If D1' "crashes", then the cooperation of D1 is required to reestablish D1' and to synchronize it with D1. Two solutions are possible. First, only resynchronize on a planned schedule independent of all crashes. Second, treat a crash as a compromise of the system, allow it, but report it to the security manager for action. This latter solution figures prominently in later approaches. Note that a compromise of the system is defined to be the possible transmission of a single bit of information, when some observable event (like a disk crash) occurs. Such compromise may well be unavoidable, and a system which is secure except for a compromise in this sense may still be very useful. First, no system is immune from the occurence of observable events, and second, since the events are observable, we know when they have occurred and can investigate them individually by agents external to the system.

Approach A generalizes in a straightforward way to an arbitrary digraph S. Each si still has its own computer and its own shadow disk for all lower levels. For example, in the S shown below, there will be n(n+1)/2 total disks. This

sn -->...--> s2 --> s1

cost is of course a major drawback to this approach. Another major drawback is the cost of reconfiguring the system.

The major advantage of the solution is the altogether trivial "proof" that the system is secure and its ability to use arbitrary system software at each level. Each level has its computational power supplied by software but turns to hardware for enforcing its security.

4. Two More Approaches

Despite the security advantages of the previous solution, it is simply too expensive for most situations because of the large number of disks required. We will show in this section how to solve the *-property problem without this cost. The savings has its own price, however. We will need to rely on more hardware devices; the user must weigh this against the cost of the first approach. It appears that there is a tradeoff between what must be trusted and system cost. This seems like a very natural approach to security. Not all environments are the same nor do they require identical solutions. A spectrum of solutions may be helpful to system planners.

To get the basic idea of this kind of solution, let D1' be the shadow of D1 as described in the previous section. If s2,...,sn wish to read any of s1's files, they can use a queuing strategy to multiplex D1'.

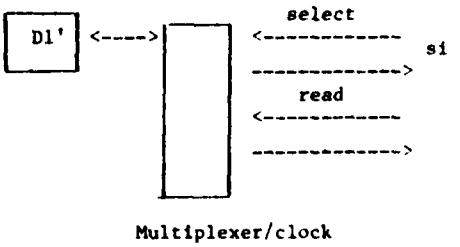


Figure 3. Multiplexing the Shadow Disk

A user in level i can select a record of D1' and receive it on the read line. The security of the system depends on the disk scheduler. Let us assume that the scheduler is a hardware multiplexer which uses a clock pulse to schedule both the disk and the disk controller round-robin. The inherent fairness of round-robin scheduling strategies insures the security of this scheme. Users with especially sensitive requirements may

want to fine tune the scheduling parameters to eliminate order-of-arrival dependencies.

We could be satisfied with this approach if we wanted to trust the disk controller. This is, after all, the minimal amount of software which must be trusted in the verification oriented approaches.

We take the solution a step further by eliminating our reliance on the security of the disk controller; in this solution we make no assumptions whatsoever about the disk controllers -- we certainly do not trust them! Figure 4 is the configuration of Approach B.

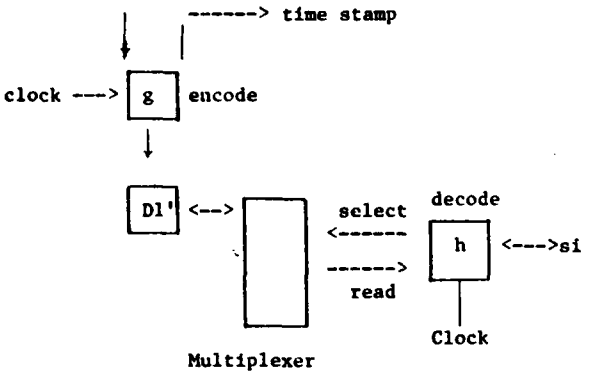


Figure 4. Approach B

The encrypting and decrypting functions may be carried out by DES boxes, public-key systems, or any of a variety of less stringent algorithms [8]. They must be trusted. The main operations defined in the system are write operations to a local disk and read operations on a disk belonging to another level. The essential idea here will be the "signing" of records by means of an encryption operation which couples the record to a unique key called a time stamp. There are many possibilities for time stamps, but the important requirement is that time stamps must constitute a nonrepeating sequence, say, 0,1,2,... . The encoding function g is determined by the key for level i (which for the moment we will take to be i) and maps a record p and time stamp t to a *message* q. That is,

$$g(p,t) = q.$$

The decoding function h is the inverse of g:

$$(p,t) = h(q).$$

To write p with key[+] k, it is necessary to execute the following protocol.

```
write (p,k)
 t <-- next time stamp;
 q <-- g(p,t);
 send (k,q) to disk;
 broadcast (k,t) over all one-way links;
```

To read record k from the shadow disk D1', a level must execute the following protocol.

```
read (k,t)
 q <-- select (k);
 (p',t') <-- h(q)
 if t=t' then return p' else SOUND ALARM;
```

The protocols simply state that sl encrypts its record using a time stamp, while si sends a request to D1' which it decrypts using h; if the signature is incorrect then the request is illegal and an alarm sounds. To see why these protocols are correct, let us prove that information cannot flow illegally from si to sj. Information can flow to the disk controller. When the controller receives a request, it can fail to respond, but this is observable and an alarm can be sounded. If the controller selects the correct record, then there is nothing to be done. If the controller selects any other record, an alarm will sound since the read protocol will fail to verify the signature. As in the case of Approach A, there is a one-bit compromise inherent in this scheme, but it is always associated with an external event.

There is no software to be trusted in this scheme; even the time stamp generator can be eliminated by using the round-robin scheduler to generate the correct time for the time stamp. Now, if si requests a record using the wrong time stamp, it will (at worst) have the request denied.

Approach B uses N disks. By adding one more time stamp to the signature, it is possible to use a single disk for all levels. The read protocol is much the same as for Approach B:

[+]That is, a record key, a public name by which the record is known. If for example p is a page, then k is the page number. The variable k is reserved for record keys. When encryption keys are called for, they will be denoted by i,j,i',j*, etc.

```
read (k,t)
 q <-- select (k);
 (p',t',t") <-- h(q);
 if t=t' then return p' else SOUND ALARM:
```

The write protocol, however, is more complicated.

It is easier to visualize the write protocol if Figure 4 is modified as shown in Figure 5.

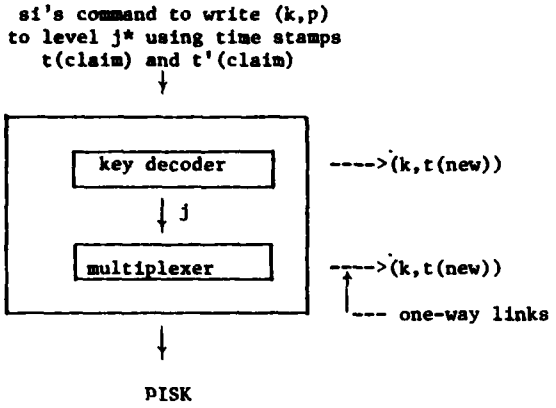


Figure 5. Single Disk Solution

In order for si to be able to complete the write, i must be able to compose the following list of arguments.

1. p, the record to be written,
2. k, the record key
3. t(claim), the time stamp which i claims will allow the page frame to be "pulled" by select(k),
4. t'(claim), the time stamp which i claims will allow p to be written into the frame retrieved and verified using t(claim),
5. j*, the "public" name of sj.

The write protocol is specified by two protocols,

can-write and do-write:

```
can-write (k,t(claim),t'(claim),j*)
 j <-- h(j*);
 q <-- select(k);
 (p(old),t,t') <-- h(q);
 if t = t(claim) and h'(t') = t'(claim) then
     exit can-write else
     SOUND ALARM;
```

```
do-write (k,p)
 t(new) <-- next time stamp
 q(new) <-- g(p,t(new),g'(t(new)));
 send (k,q(new)) to disk;
 broadcast (k,t(new)) over one-way links;
```

The encoding and decoding functions g' and h' above are the functions determined by the cryptosystem with key j (g and h are still reserved for key i). Since only those levels which receive the original time stamp for k will be able to match the second time stamp, the effect of the do-write is to "mask" portions of the disk file from unauthorized levels. To write record p using key k, a user at level i first composes a mask that only appropriate levels j may find; the mask contains the read time stamp which functions as in the previous case, but it also contains a second entry: the encryption of the time stamp using g'. With the triple thus prepared, it is encrypted and readied for broadcast down all one-way links, but in order to insure that the record key only finds its way to files with a matching record mask, the can-write protocol examines the destination triple. If the controller has attempted to show i an illegal destination the read stamp or the write stamp will not match (unless the "name" of j has been compromised -- we deal with this difficulty, in a moment) and an alarm will sound. The read protocol functions as before.

In most implementations of this scheme, it will be necessary to pass both i and j as parameters to the protocols. Since this is a possible path of information flow, it is necessary to encrypt i and j. The actual keys i and j are really generated by an encryption function g(x,y), where x is a private key and y is the public name of the level. So if level j is known by the name j* and retains the private key AABBCC, then we set

$$j = g(AABBCC, j^*).$$

5. Final Comments

The key to the solutions to the *-property problem given above is to pack as much as possible into physical security. Software solutions also need, in addition to trusted software, a variety of trusted hardware devices, so our approaches simply shift the starting points of the solutions.

Aside from the previously noted problems -- the lack of systems flexibility, degradation of disk speeds (2:1 on local disks, N:1 on remaining disks) -- there is much to recommend these solutions. They avoid completely the denial of service issue and can be adapted to treat, for instance, the confinement problem [8]. They provide a tradeoff between cost and system security. Most importantly, they allow any software whatsoever at the computers corresponding to the levels. For advanced military and other applications in which multilevel security is but a component of broadly based networking or distributed facilities, the heterogeneity of the resulting system is a paramount design requirement. Current software approaches fly in the face of this requirement, since they require trusted -- in most cases, uniform -- software at the nodes of the network.

We anticipate that more complex protocols can be devised to deal with a variety of problems using only physical properties of the system. For example, similar techniques can be used to deal with selective downgrading, device sharing (e.g., sharing of secure printers), and confinement. We will present these designs elsewhere. As we have already argued, these may be feasible approaches to "verified" security systems.

Acknowledgements

The outlines of the designs sketched in this paper were obtained at the Draper Laboratory Summer School on Computer Security in July 1979. We would like to thank Bart DeWolfe and Anne Marmour-Squires for inviting us to participate in the session on verification-oriented approaches to security. Of course, we also owe a great deal to the other participants in that session; it was their aggressive response to our skepticism regarding traditional verification/kernel approaches that led to these ideas.

References

[1] DeMillo, R., D. Dobkin, A. Jones, R. Lipton, Foundations of Secure Computation, Academic Press, 1978.

[2] Bell, D., L. LaPadula, "Secure Computer Systems: A Mathematical Model," Mitre Corporation, MTR-2547, Vol. 2, November, 1973.

[3] Parker, D., Crime by Computer, Simon and Schuster, 1975.

[4] Newmann, P.G., R. Fabry, K. Levitt, L. Robinson, J. Wensley, "On the Design of a Provably Secure Operating System, " Proceedings IRIA International Workshop on Protection in Operating Systems, 1974, pp. 161-176.

[5] DeWolf, J. Barton, editor, "Final Report of the 1979 Summer Study on Airforce Computer Security, R-1326, Charles Stark Draper Laboratories, October, 1979."

[6] DeMillo, R., R. Lipton, A. Perlis, "Social Processes and Proofs of Theorems and Programs," Communications of the ACM, Vol. 22, No. 5. (May, 1979), pp. 271-280.

[7] Denning, D., P. Denning, "Data Security," ACM Computing Surveys, September, 1979, pp. 227-250.

[8] Lampson, B., "A Note on the Confinement Problem," Communications of the ACM, Vol. 16, No. 10, (October, 1973), pp. 613-615.